



LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

National Security Forensics

B. M. Ng, N. C. Perry

October 13, 2011

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.



National Security Forensics
FY2011 END-OF-YEAR REPORT

B.M. NG AND N.C. PERRY

Lawrence Livermore National Laboratory
Livermore, CA

Document ID: LLNL-TR-505392
Contact: Brenda Ng, 925-422-4553

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

Auspices Statement

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

**NATIONAL SECURITY FORENSICS
FY2011 END-OF-YEAR REPORT**

B.M. Ng and N.C. Perry

DESCRIPTION

The objective of this effort is to research and develop inversion methods for mapping observations to the input parameters of the underlying physical system that could have generated these observations. This report describes our FY11 progress in developing a surrogate model of the forward model, using this surrogate forward model to develop inversion capabilities, and partitioning the input-output space to facilitate assessment and implementation of our inversion capabilities, as well as adaptively propose new computer code runs to enhance our training data. This work was funded by NA22/SAM in the NN2001-06 area, under project number LL10-Forensics-PD06.

TABLE OF CONTENTS

1 Introduction	3
2 Data.....	4
3 Methods.....	4
3.1 Forward Mapping	5
3.2 Inverse Mapping.....	8
3.3 Viable Input-Output Space	11
4 Results	15
4.1 Forward Mapping	15
4.2 Inverse Mapping.....	15
4.3 Viable Input-Output Space	16
5 Discussion & Future Work.....	16
References	17

PROJECT NARRATIVE**1 INTRODUCTION**

The objective of this effort is to research and develop inversion methods for mapping observations to the input parameters of the underlying physical system that could have generated these observations. In this report, we will use the terms “observation” and “output” interchangeably.

Generally speaking, the inverse problem is to find the inverse of the forward model. From the forward model, one can predict observations from input parameters. In our project, the forward model is embodied within complex nonlinear computer codes that are treated (for this analysis) as “black box” representations of the physical system. While we can run the computer codes to generate “points” in the input-output space (each point would be a tuple of input parameter values and their corresponding observation parameter values), our particular forward model cannot be analytically expressed, much less inverted mathematically, in exact forms.

Theoretically, one can *approximate* the forward model by running the codes multiple times to generate many points in the input-output space, then fit a *surrogate* model to capture the input-output relationship reflected in these points. This surrogate model would be used to enable accurate estimation of the observation parameters when their values are known only at a finite number of points. However, since each computer run is expensive, generating points in the input-output space would require judicious choice of the input parameters. Choosing which inputs to run the forward model to maximize gain of information about the mapping (from inputs to observations) is key.

In the first year (FY11), the project goal is to deliver a first-cut version of the inversion framework. We have taken the approach of:

1. developing a surrogate model of the forward model
2. using the surrogate forward model to develop inversion capabilities
3. partitioning the input-output space in such a way that supports inversion and allows for intelligent proposal of new computer code runs (in order to improve the surrogate forward model and subsequently the inverse model)

Additionally, we have made progress on characterizing the coverage of the viable input-output space, which is necessary to improve the speed and completeness of the assessments that will be made by the end-user of these tools. Not all inputs yield viable outputs for the physical system, so analogously, not all inputs will result in successful runs by the codes. Being able to know a priori which regions are viable can guide effective exploration of the input-output space without wasting time initiating costly runs using inputs from non-viable regions.

Section 2 describes the data available to us during our first-year exercise. Section 3 describes our efforts in the development of the surrogate forward model, the inverse model, and strategies for viable space coverage. Section 4 describes empirical results. Lastly, Section 5 summarizes future work.

2 DATA

The data are generated from a complex nonlinear computer code. While there are many input and output parameters, we restricted this year's scope to those pertinent to a small part of domain space. More details on the computer code, domain space, and observable data are listed in an annex to this document (LLNL document COPJ-2011-0541).

In our first year, we focused on four inputs and four outputs. The four inputs were chosen based on their orthogonality (i.e., the input parameters reflect different aspects about the physical system) and the four outputs were chosen based on expert knowledge. In future years, we plan to expand our data beyond one subclass, as well as examine other inputs and outputs. These planned efforts are outlined in Section 5.

3 METHODS

As alluded above, the data set consists of tuples of input and output parameters. Each tuple constitutes a point in the input-output space and is the result of a single run that requires significant computer resources to complete. Thus, populating the entire input-output space by exhaustively running the codes on all possible instantiations of the input parameters would be prohibitively expensive.

Our first step was to develop a surrogate model that can capture the forward model.

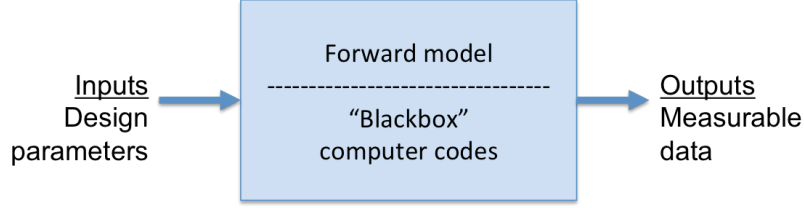


Figure 1. Our forward model is encapsulated in complex, nonlinear computer codes.

Once achieved, we can apply this surrogate model to cheaply populate the rest of the input-output space, as well as to summarize our state of knowledge about the input-output mapping.

One might ask “why not develop a surrogate model directly on the inverse model?” While it might seem like extra work to develop the surrogate model on the forward model then invert on the surrogate, there are two main reasons why this approach is better for long-term development. The primary reason is that, if we want to improve the predictive power of the surrogate forward model over a particular region in the input-output space, we can easily add more points to that region by running our codes using the input values that span that region. If we were to do the same for the surrogate inverse model, this would be difficult because we do not have codes that would generate the input values based on the output values. (In fact, this is precisely the capability that we are trying to develop.) The secondary reason is that our subject matter experts (SMEs), who developed and are maintaining the codes, have in-depth knowledge of the forward model (which is a “black box” to us but not to them). Their SME insights, in the form of functional relationships between groups of input/output variables, can be elicited and incorporated to improve our surrogate forward model. Again, this will be difficult if we are developing the surrogate inverse model directly because the SMEs can share more about the forward model than the inverse model. In sum, the knowledge elicitation is easier in the forward direction than the inverse direction. The underlying mathematical theories behind the model development, however, are still transferable so we can leverage the same algorithms for both the forward and inverse models.

Before we proceed, we introduce some notation. Let $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$ denote the data where $\mathbf{x}_i \in \mathbb{R}^4$ and $\mathbf{y}_i \in \mathbb{R}^4$. The forward model f is a mapping from inputs to outputs as follows: $\mathbf{y}_i = f(\mathbf{x}_i)$.

3.1 Forward Mapping

For this task, we first implemented an inverse distance weighting method, known as the Shepard’s method, for multivariate interpolation between \mathbf{x}_i to \mathbf{y}_i . In any inverse distance weighting method, the *interpolating function* \hat{f} is constructed as an approximation to f , which takes on the general form of:

$$\hat{\mathbf{y}} = \hat{f}(\mathbf{x}) = \sum_{i=1}^N \frac{w_i \mathbf{y}_i}{\sum_{j=1}^N w_j}$$

Given the test input \mathbf{x} , the interpolated output $\hat{\mathbf{y}}$ (distinct from the actual output \mathbf{y}) is computed as a weighted sum of the “training” output values $\{\mathbf{y}_i\}_{i=1}^N$ which were derived from the data. The weights in turn are computed from the test input \mathbf{x} and the “training” inputs $\{\mathbf{x}_i\}_{i=1}^N$, which the Shepard’s Method [Shepard, 1968] defines as:

$$w_i = \frac{1}{d(\mathbf{x}, \mathbf{x}_i)^p}$$

$d(\mathbf{x}, \mathbf{x}_i)^p$ denotes the distance from \mathbf{x} to \mathbf{x}_i raised to the p -th power. Thus, the weight increases the distance decreases. The parameter p affects the degree of smoothing in the interpolation. Higher p values will weight contributions from nearby points more heavily.

There is also a modified version of the Shepard's Method [Franke & Nielson, 1980], which uses only the nearest outputs within a R-sphere for interpolation. The weights in this case are given as:

$$w_i = \left(\frac{R - d(\mathbf{x}, \mathbf{x}_i)}{Rd(\mathbf{x}, \mathbf{x}_i)} \right)^2$$

R is defined as the maximal distance, $R = \max_i \{d(\mathbf{x}, \mathbf{x}_i)\}$. The modified Shepard's Method imposes symmetric weights around each training points and has been found to outperform the original Shepard's Method in practice. This was also validated in our effort so we decided to pursue improvements to the modified Shepard's Method.

A challenge in our data is that the input variables are of different magnitudes thus resulting in distances that are greater than two orders of magnitude in some cases. In essence, the dimension with the largest magnitude was ignored in the interpolation function. To mitigate this, we normalized the input variables to within 0 and 1.

Additionally, since our input-output mapping is multi-dimensional and highly nonlinear, far away points are virtually of no use in providing information about the test point. Thus, we restricted the interpolation to a set of local "grid" points (whose input values enclose the training point's input) for improved interpolation accuracy and speed. The assignment of grid points to a particular test point was achieved through partitioning the input space into disjoint *polytopes*¹. Figure 2 shows a two-dimensional representation of this partitioning.

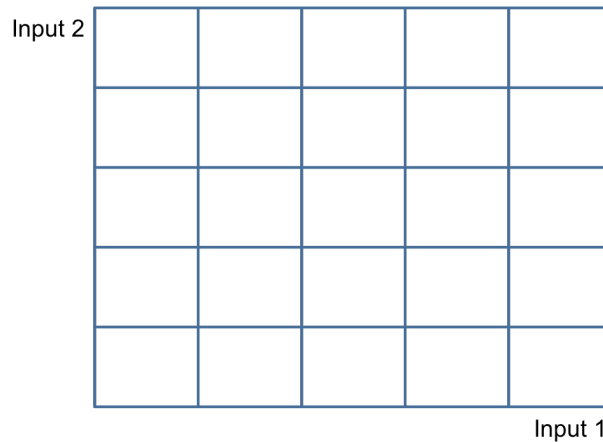


Figure 2. Partition of a two-dimensional input space.

For simplicity, we have chosen to work with hypercube-like polytopes. In four dimensions, each of these polytopes has 2^4 or 16 vertices. (For further enhancement to interpolation accuracy, we can go beyond just the vertices, but choose additional points within or at the surface of the polytopes. Such a tradeoff study of improved accuracy as a function of additional points is in our plans for future work.) The general idea for interpolating the output value of a given test point is as follows:

¹ A polytope is a geometric object with flat sides that can exist in any number of dimensions. A 2-dimensional polytope is a polygon and a 3-dimensional polytope is a polyhedron.

1. We first lay the ground work by partitioning the input space into disjoint polytopes. We apply the computer codes to run the input values given by the polytopes' vertices. The resulting data becomes the “database” of points on which our interpolation-based surrogate forward model is based. We do this only once unless we find that the database of points yields poor interpolation performance.

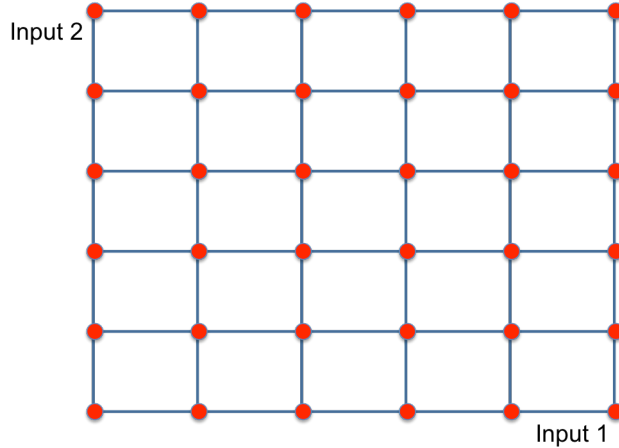


Figure 3a. The partition defines the points (in red) to be run by the computer codes.

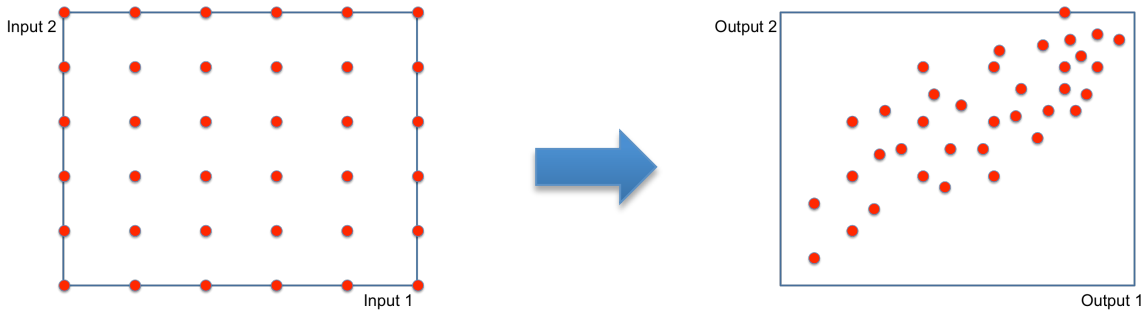


Figure 3b. Each computer run will yield a corresponding point in the output space for each point in the input space. The set of inputs and outputs constitutes the training data.

2. Given a test input, our goal is to use our surrogate forward model to infer its output. We first look up the (unique) polytope encloses the test point's input. Then we apply the modified Shepard's method on this polytope's points *only*.

$$\hat{y} = \hat{f}(x) = \sum_{i \in \mathcal{P}} \frac{w_i y_i}{\sum_{j \in \mathcal{P}} w_j}$$

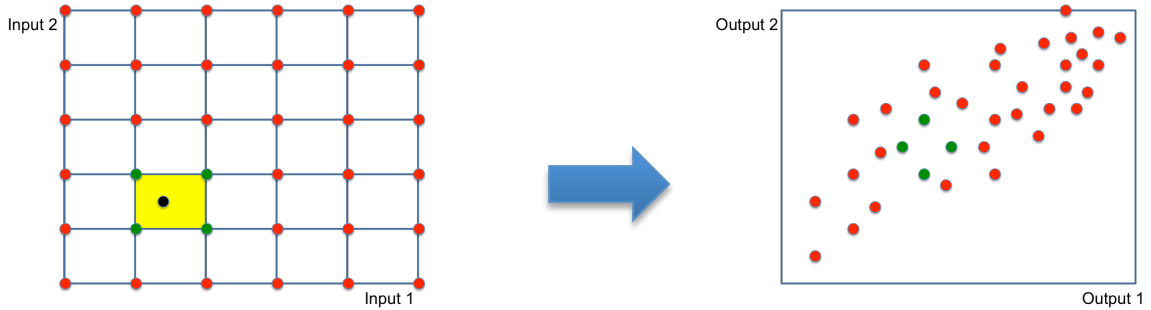


Figure 4. Given the test point, whose input is represented as a black point within the yellow rectangle, only the green points which comprise the corners of the rectangle, will be used in the interpolation.

This is distinct from the traditional application of the modified Shepard's method, which bases its interpolation on the entire set of training points. Our variant of the modified Shepard's method restricts the interpolation to the polytope's points, which improves speed as well as accuracy in our specific multi-dimensional, highly-nonlinear problem.

An additional advantage to partitioning the input space is to be better able to characterize regions of high nonlinearity or sensitivity, which would require more training points and/or sophisticated models with higher degrees of freedom, for improved approximations.

Input 2	25%	28%	30%	25%	12%
	20%	22%	25%	20%	15%
	15%	15%	18%	18%	15%
	12%	13%	15%	16%	17%
	10%	15%	15%	10%	10%
Input 1					

Figure 5. Partition of a two-dimensional input space with confidence of output estimation assigned to each polytope.

3.2 Inverse Mapping

In the inverse direction, from outputs to inputs, the polytope-partition-based interpolation scheme developed for the forward direction does not readily apply. The reason is as follows: If we partition the output space into polytopes and wish to use the polytope vertices as the training data, there will be no direct way for us to get the corresponding input values because our codes only go in the forward direction, i.e., the codes take inputs and return outputs.

Alternatively, we could try to choose the input values that, when run by the codes, will return output values that are in a quasi-grid formation. Again, to be able to choose such input values would require us to have already solved the inversion problem, which is the very thing we are trying to address.

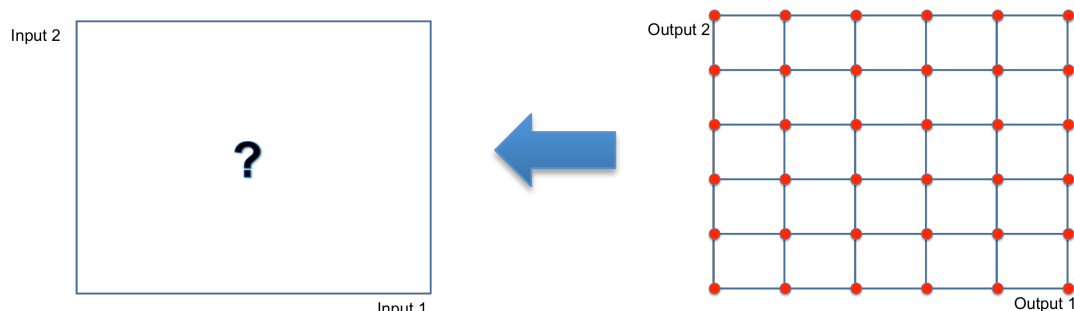


Figure 6. If we partition the output space via a regular grid, it would be difficult to obtain training data because we have no easy means of recovering the inputs that correspond to the outputs.

Moreover, it would be of added benefit if we can utilize the training data that was already established for the polytopes in the surrogate forward model, instead of having to establish a new set of training data for the inverse direction.

With this consideration, we realized that, in order to make our variant of the modified Shepard's Method work in the inverse direction, all we really needed was to enclose our test output in some closed region and to be able to find this region's image in the input space. Then, we would be able to proceed as before by restricting our interpolation to points belonging to this region.

What this means is that we do not need polytope partitioning in the output space. Instead, we can use the training data established for the polytopes in the surrogate forward model to define regions over the output space. For a given polytope in the input space, its vertices (if they represent valid inputs in the physical system) will map to points in the output space. These points in the output space can be used to create an n -dimensional convex hull².

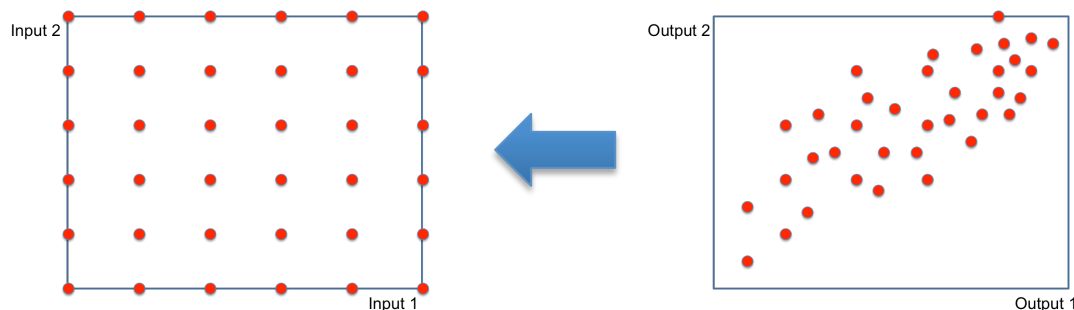


Figure 7a. The data generated for the surrogate forward model can be used for the inverse model...

² A convex hull for a set of points X in a real vector space V is the minimal convex set containing X . When the set X is two-dimensional, imagine stretching a rubber band to enclose the entire set X then releasing it; the region defined by the taut rubber band is the convex hull of X . In this year's work, we are dealing with four-dimensional convex hulls because our output space is four-dimensional.

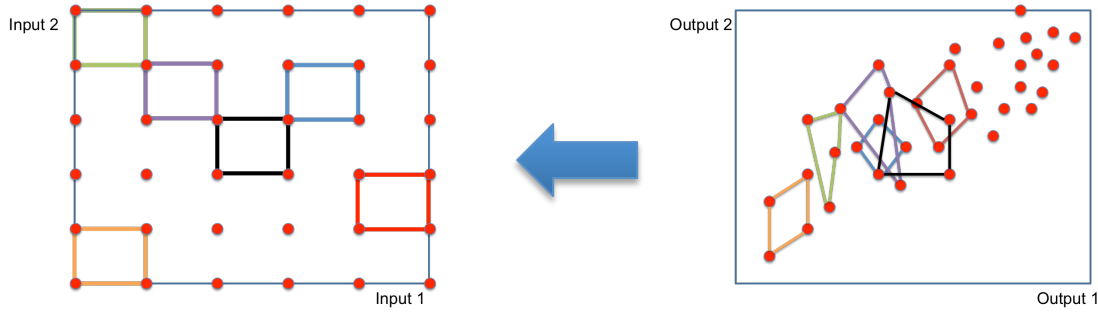


Figure 7b. ... so long as we decompose the output space into convex hulls that correspond to the polytopes from the partition in the input space. (Only select convex hulls and their polytopes are shown.)

Unlike the input-space polytopes, the output-space convex hulls could be overlapping. This means, for a given test output, there might be multiple convex hulls that enclose it. With this caveat, we are now ready to proceed as in the forward case.

1. For a given test output, the goal is to find its corresponding output. We first examine where this test output lies and return any convex hull that encloses the test output. For each of these convex hulls, we apply the modified Shepard's method on the vertices of the convex hull. In the equation below, we denote the k -th convex hull as \mathcal{C}_k .

$$\hat{\mathbf{x}}_k = \hat{f}_k^{\text{inv}}(\mathbf{y}) = \sum_{i \in \mathcal{C}_k} \frac{w_i \mathbf{x}_i}{\sum_{j \in \mathcal{C}_k} w_j}$$

$$w_i = \left(\frac{R - d(\mathbf{y}, \mathbf{y}_i)}{R d(\mathbf{y}, \mathbf{y}_i)} \right)^2 \text{ with } R = \max_i \{d(\mathbf{y}, \mathbf{y}_i)\}$$

2. If there are K convex hulls that enclose the test output, then we will have K estimates of the input parameters, $\{\hat{\mathbf{x}}_k\}_{k=1}^K$. Since different input values can lead to the same output values in the physical system (i.e., the forward model is not one-to-one), these K estimates should not be averaged but returned as a histogram to convey a distribution over potential inputs that could have generated the test output. (This is part of near-future work – currently, we utilize all the points from the K convex hulls to compute one single estimate for the inputs. We do not yet compute a separate estimate for each of the K convex hulls and perform a histogram on the estimates' values.)

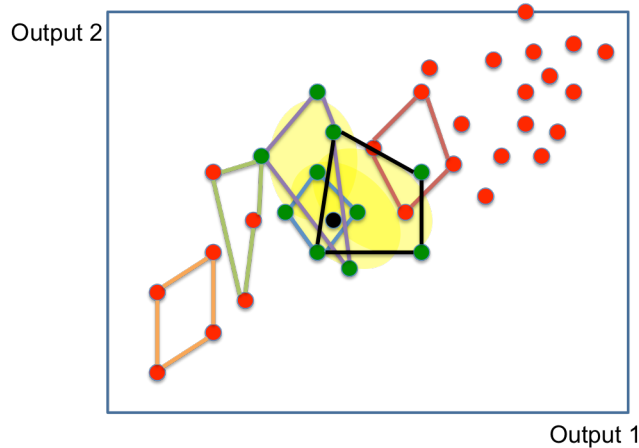


Figure 8. In this example, three convex hulls (with purple, blue and black edges; region highlighted in yellow) enclose the test output (in black). Thus, the vertices of these three convex hulls (in green), as well as their images in the input space, are used in computing the inverse of the test output.

3.3 Viable Input-Output Space

So far, we have not discussed the fact that not all inputs map to outputs. Some input values represent situations that are not inadmissible in practice. Hence, when the codes are set with these inputs, failed runs result because the inputs are outside of allowable parameter values.

Returning to our discussion of the surrogate forward model, imagine that, if we had unknowingly partitioned over some non-viable space, then we might have at least one polytope with vertices that do not have corresponding outputs. A test input assigned to this polytope will have fewer points for its interpolation of the output, because we can only use viable points (i.e., those with inputs and outputs) in the equation for the Shepard's Method. If many vertices in this polytope are non-viable, we might run into *extrapolation* instead of *interpolation*, and estimation performance could quickly degrade.

An alternative is to replace each non-viable point with a viable point within the same vicinity, so we are still using the same number of points for interpolation. In effect, we are moving the boundary of the polytope inward to encompass only the viable space. This boundary shift would likely cause the polytope to no longer be a polytope, but a higher-dimensional convex hull. But since we have already gained experience working with convex hulls in the inverse model, this approach is doable.

To summarize, this part of the effort is focused on identifying the viable input-output space, so that we can efficiently populate more points in this space to ensure that we are interpolating with a minimally required number of training points for each test point. This in turn improves interpolation estimates, which will improve the accuracy of the surrogate forward model and the inverse model.

A failed run takes significantly less time than a successful run, so we decided on the approach of closing in on the viable space from points that are in the non-viable space. We make the assumption that, for our particular subclass, there is only one contiguous viable space and there is also only one contiguous non-viable space. Our approach to identify and populate the viable space is as follows:

1. For each input variable, we elicit (from the SME) a minimum value and a maximum value that are outside of that input variable's admissible range of values.
2. Using these min/max values for the input variables, we can define a polytope in the input space.

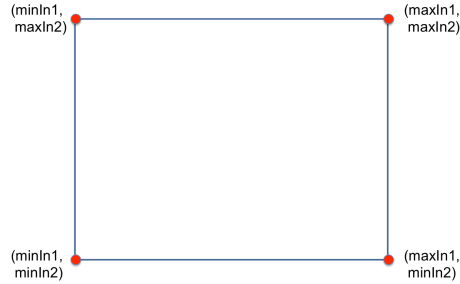


Figure 9. A polytope is defined in the input space based on the minimum and maximum values of each input. This figure is a two-dimensional abstraction of our four-dimensional input space. “minIn N ” and “maxIn N ” denotes the respective minimum and maximum values of the input N .

3. We compute the centroid, or elicit (from the SME) a user-specific centroid, within this polytope. We run the code on the centroid to ensure that it defines a viable point.

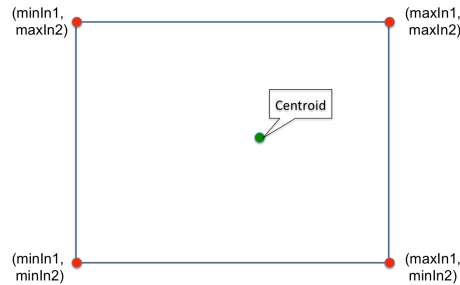


Figure 10. The user-defined centroid does not need to be the same as the mathematically defined centroid of the polytope. The centroid must be a viable point.

4. For each of the 16 vertices in the polytope, we form a vector from the vertex to the centroid.

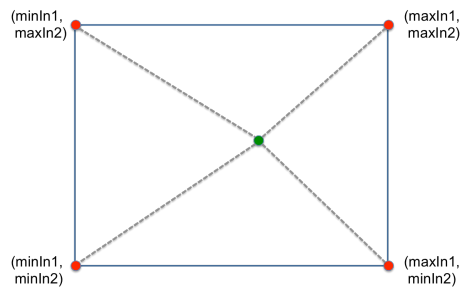


Figure 11. Here, we show a two-dimensional perspective depicting four “vertex-to-centroid” vectors. Our 4-dimensional polytope has 16 such vectors in total.³

5. For each of the 16 vertex-to-centroid vectors, we incrementally “move in” along this vector and sequentially run the code on new inputs that lie on this vector until the code returns a successful run. The input/output tuple corresponding to the successful run now defines a viable point in the space.

³ For a general n -dimensional cube, the number of “ k -cells” is $\binom{n!}{k!(n-k)!} \times 2^{n-k}$. A 0-cell is a vertex; a 1-cell is an edge; a 2-cell is a face, etc. Our 4-dimensional polytope has 16 vertices, 32 edges and 24 faces.

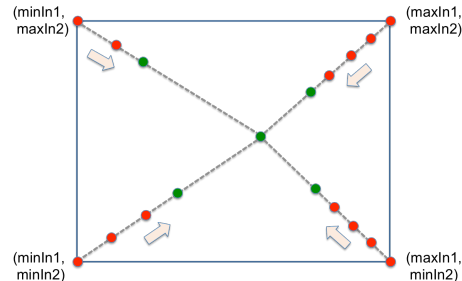


Figure 12. Inputs along these “vertex-to-centroid vectors” are explored sequentially until a viable point is found.

6. Once a viable point is achieved on each of the 16 vertex-to-centroid vectors, we would have 16 viable points on which we can define a convex hull of potential viable space.

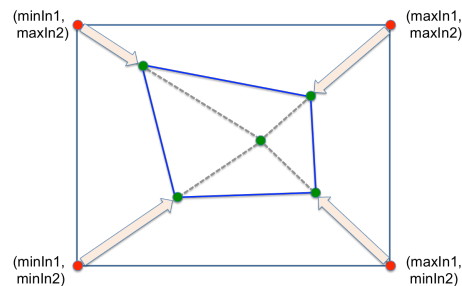


Figure 13. The result of this procedure is a tighter “bounding box” in the form of a convex hull that encloses the estimated viable region.

To gain a more faithful coverage of the viable space, we also checked for viable points near the boundary of this convex hull, as follows:

7. For each edge of the convex hull, we trisect the edge into three same-length segments via two vectors from centroid to the edge.

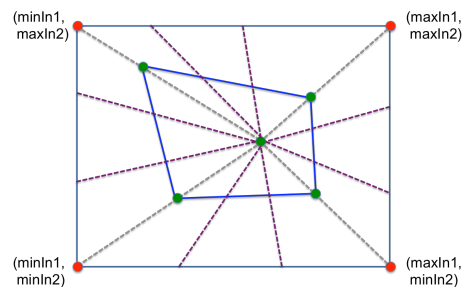


Figure 14. Each edge of the convex hull is trisected. Our 4-dimensional polytope has 64 such “edge-to-centroid” vectors in total.⁴

8. The intersection between the edge and the vector specifies an input point. We run the code on this input point to check for its viability.

⁴ Our 4-dimensional polytope has 32 edges (see footnote [3]). For each edge, we create two “edge-to-centroid” vectors, so there are $32 \times 2 = 64$ such vectors.

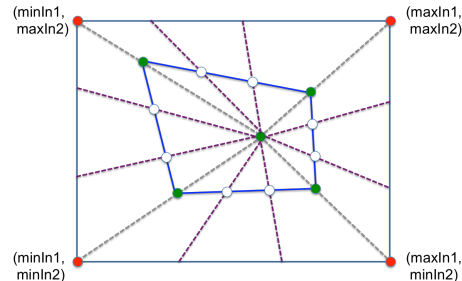


Figure 15. The trisection points are the starting point for the viable space exploration. We apply the inputs at these points to the computer runs first.

9. If the input point is viable, then we incrementally “move out” along the vector and sequentially run the code on new inputs that lie on this vector until the code returns an unsuccessful run. Otherwise, if the input point is not viable, then we incrementally “move in” along the vector until the code returns a successful run. The input/output tuple corresponding to the last successful run now defines a viable point and replaces the original input point on the convex hull edge.

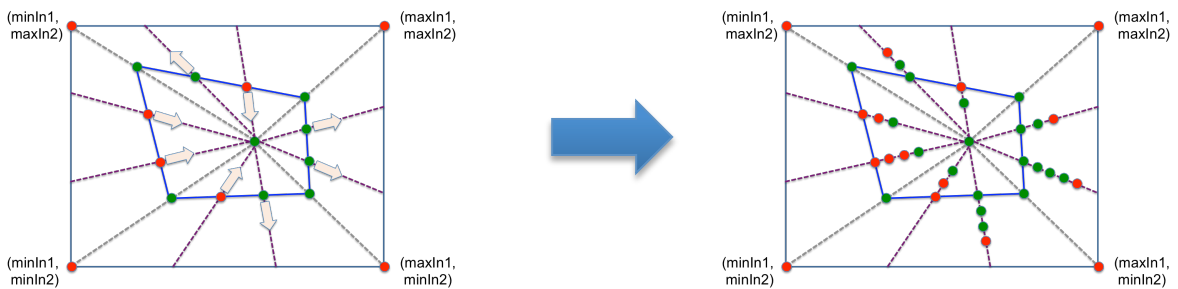


Figure 16. We explore inward if the trisection points are non-viable, and explore outward if the trisection points are viable. This strategy ensures we are closing in on the boundary of the viable region.

10. Once this process is completed for all the “edge-to-centroid” vectors, the boundary of the viable space might no longer be a convex hull.

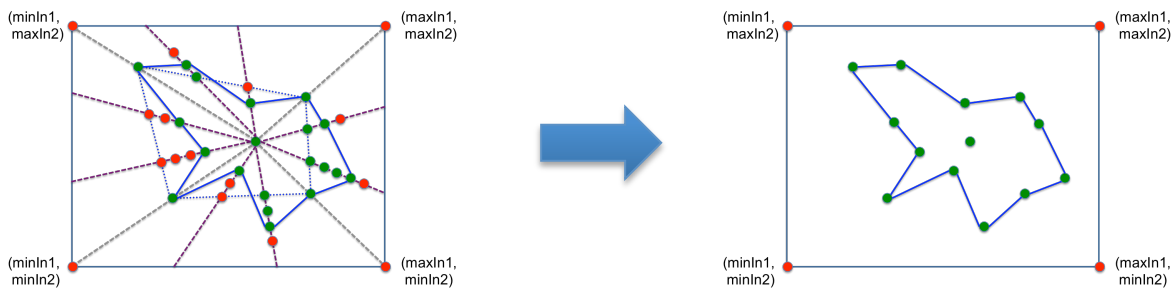


Figure 17. At the end of this procedure, we get an even tighter “bounding shape” that encloses the estimated viable region. However, this shape might no longer be a convex hull.

To further improve coverage of the viable space, we have also repeated the above procedure for vectors that emanate from the centroid to (quasi-equidistant) points on the faces of the resulting region (that encloses the viable space). Currently, we restrict the search to four vectors per face.

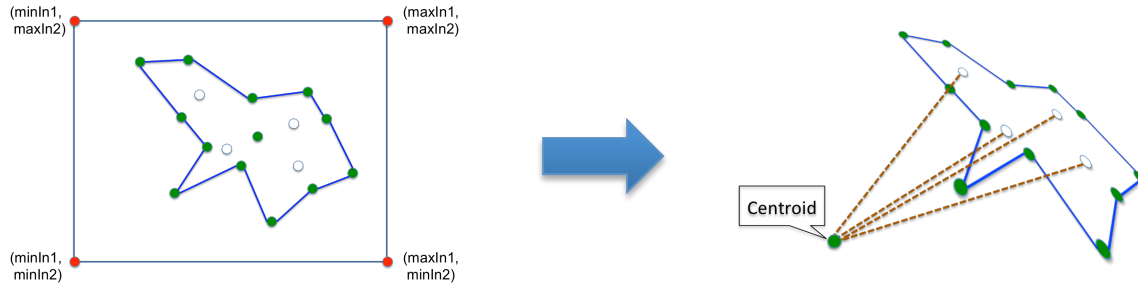


Figure 18. Additionally, we repeat this exploration procedure along each set of four vectors that emanates from the centroid to the face. Our 4-dimensional polytope has 96 such “face-to-centroid” vectors in total.⁵

4 RESULTS

We have evaluated our surrogate forward model, our inverse model and solutions for the coverage problem and the results are presented in this section.

4.1 Forward Mapping

We evaluated three methods:

1. the traditional application of the modified Shepard’s Method which utilizes all the training points
2. the method in (1) with normalized inputs
3. the method in (2) using only the points from the polytope assigned to the test input

We inferred each output separately, as well as all outputs at once.

The training data were generated from using the polytope vertices (as inputs to the codes) and the test data were generated from points within the polytopes, so the training and test data sets are completely disjoint. The training data contains approximately the same number of points as the test data. (More detail is available in the LLNL document COPJ-2011-0541.) Only the inputs from the test data are used in the methods. The outputs from the test data are used as “ground truth” to compute the error between the inferred output values from the actual output values. The following table shows the median error, for separate output parameters as well as the aggregate, across all three methods described.

Output(s)	Modified Shepard’s Method	w/ normalization	w/ normalization and using only polytope points
1 only	1700%	1700%	14%
2 only	3800%	1400%	21%
3 only	34%	8%	11%
4 only	74%	34%	17%
All	74%	34%	17%

Our developed method, as shown in the last column, reflects the best performance since it exploits locality in its inference of the output parameters and has the fastest runtime because it uses less points for interpolation.

4.2 Inverse Mapping

As before, we evaluated three methods:

1. the traditional application of the modified Shepard’s Method which utilizes all the training points

⁵ Our 4-dimensional polytope has 24 faces (see footnote [3]). For each face, we create four “face-to-centroid” vectors, so there are $24 \times 4 = 96$ such vectors.

2. the method in (1) with normalized outputs and using the training points from the multiple convex hulls that enclose the test output
3. the method in (1) with normalized outputs and using the single convex hull that is the image of the polytope that encloses the true input of the test point

In practice, method 3 is not really possible since we would not know which convex hull is the image of the polytope that encloses the true input. So it is provided as an estimate of a best-case result using this method. Like before, we inferred each output separately, as well as all outputs at once.

For the inversion, the same training and test data sets for evaluating the forward model were used. As before, the training data contains approximately the same number of points as the test data; the points are distinct. (More detail is available in the LLNL document COPJ-2011-0541.) In contrast to the forward model evaluation, only the *outputs* from the test data are used in the methods. The inputs from the test data are used as “ground truth” to compute the error between the inferred input values from the actual input values. The following table shows the median error, for separate output as well as the aggregate, across all three methods described.

Input(s)	Modified Shepard’s Method in the inverse direction	w/ normalization and using points from all the convex hulls the enclose the test output	w/ normalization and using only the single convex hull that encloses the test output and is also the true image of the polytope that encloses the true input
1 only	19%	17%	12%
2 only	53%	41%	27%
3 only	72%	70%	38%
4 only	49%	40%	20%
All	51%	41%	21%

4.3 Viable Input-Output Space

The viable space methods described above have been applied in the same part of domain space that we tested our inversion models on. A new set of input parameters has been defined. We plan to run our complex nonlinear computer code using these input parameters and to repeat our surrogate and inverse analysis with the new data, but this has not yet been completed.

5 DISCUSSION & FUTURE WORK

We have accomplished a first-cut version of the inversion framework. Along the way, we have developed a surrogate of the forward model and implemented methods to infer the boundary of the viable input-output space. Our decision to use a partition-based design would facilitate adaptive sampling of the viable space and help policy-makers with quickly coming up with exclusion assessments further down the pipeline.

Ideas for future work include:

1. Continue to expand the coverage of viable space by proposing more inputs for new data
2. Perform a more complete analysis of the current data, such that we can attribute confidence to each polytope or convex hull in the partition
3. Explore other surrogate modeling methods, such as kriging and Gaussian processes
4. Exploit functional relationships between system variables to improve modeling and inference

5. Further improve our application or implementation of the Shepard's Method
6. Expand analysis to more inputs and outputs
7. Expand analysis to additional subclasses in the physical domain
8. Understand the unique properties of the data, such as how the transformation of a regular grid in the input space can map to a non-disjoint grid in the output space, as well as any sparse structures that can be exploited

REFERENCES

Shepard, D., "A Two-Dimensional Interpolation Function for Irregularly-Spaced Data", *Proceedings of the 1968 ACM National Conference*, 1968, pp. 517-524.

Franke, R. and Nielson G., "Smooth Interpolation of Large Sets of Scattered Data", *International Journal for Numerical Methods in Engineering*, Vol. 15, 1980, pp. 1691-1704.

Barber, C.B., Dobkin, D.P., and Huhdanpaa, H.T., "The Quickhull algorithm for convex hulls," *ACM Trans. on Mathematical Software*, 22(4):469-483, Dec 1996, <http://www.qhull.org>